



# Software-rich Chips

September 2002



## Software-rich Chips

The challenge of designing silicon systems has primarily been a problem of designing the silicon itself, and embedded software to run on the chip has been treated as an afterthought. The relentless march of Moore's law means that embedded processors continue to become cheaper, faster and lower-power and so more and more of a system is most economically implemented in software. In turn, this leads to explosive growth in software-rich chips.

But there is a problem. In many cases the embedded software must be developed for a chip that does not yet exist. And even when the chip is available as the ultimate reference platform, the opaqueness of the internals means that it is far from ideal. A model of the chip must be built that is both accurate and with a performance high enough to allow software developers to use the model as a surrogate for the actual chip. If the model is not accurate enough, then it is not possible to derive useful performance data from the model, including whether the system will perform as expected. If the model is not fast enough then a duplicate implementation strategy will be necessary, developing the software as much as possible on a host platform and then porting it to the slow model for some level of validation.

Traditionally, the model of the system has been built in hardware: the ubiquitous single-board computer. This is costly, especially for the increasing number of systems that contain multiple processors, and does not scale well to large development groups. New technology from VaST makes it possible to create a virtual platform in software, with accuracy and speed such that software development can be done directly on the virtual platform, without needing either hardware boards or porting.

An additional benefit of this technology is that it allows system architects to explore a variety of implementations using a realistic body of software to assess the system performance to find optimum tradeoff between performance, cost and power.

VaST's technology makes software-rich chips possible.

## Semiconductor Economics Drives the Software-rich Chip Transition

Moore's law states that the number of transistors on a chip doubles every 18 months. In addition, each of those transistors is cheaper. Although it takes increasing amounts of capital to build fabs with enough capacity to make this remain true, this looks set to continue for the coming years. Looked at the other way round, Moore's law states that the cost of implementing a given function in silicon halves every 18 months. Notably, the cost of implementing a processor halves every 18 months. In addition, processors continue to get faster and lower power, adding further to the attraction of implementing silicon functionality by using processors and embedded software.

It is easy to be misled that system companies, who ship the system that is sold to the eventual consumer, want to build chips for "competitive advantage." In fact, system companies want to gain differentiation for their products in the most cost-effective way possible. Since chip design is very expensive and the schedule is very risky, designing a chip from the ground up to gain differentiation is a last resort. If possible they would prefer to use a standard or near-standard chip, and gain the differentiation through other means. Of course, there are always some systems where the differentiation comes from creative semiconductor design. The highest bandwidth routers or the highest performance graphics chips cannot be built around a standard microprocessor since still higher performance is available to a competitor who is prepared to invest in special high performance blocks. However, many systems are not like this: the required performance for a cell-phone or a 100 megabit/second Ethernet hub does not change with process technology. Instead, changes in process technology make different implementation choices possible. What is the optimum choice will depend on some combination of cost (mainly chip-size), speed and power. In most cases, this means use of processors whenever possible.

It is instructive to look at how cell-phone implementation has changed over the years. When the digital cell-phone standards such as GSM were first introduced, they were on the limit for being implemented in contemporary silicon, both in size and performance. Implementations had multiple chips using clever hardware structures. Although there was typically an embedded microprocessor for control functions, specialized functionality, such as radio equalization or encryption, was done using specially designed gate-level structures. As Moore's law advanced, there came a point that much of the functionality of these custom gate-level structures could be implemented by an on-chip digital signal processor (DSP). A typical baseband chip of this generation consisted of an embedded microprocessor and an embedded DSP along with buses and memories to connect them. There was very little custom hardware. A DSP is comparatively power-hungry, so more recently functionality has been moved back out of the DSP into hardware in order to reduce power further. This has enabled the phones that are available today, where the entire phone is smaller than the battery of the previous generation.

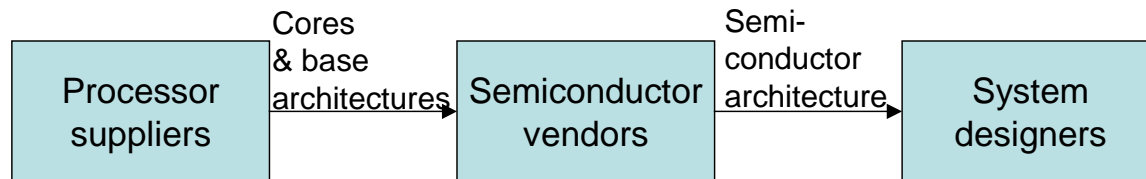
So it is Moore's law and the irresistible drive of semiconductor manufacturing economics that is driving the trend towards software-rich chips. This is reflected in industry analysis. For example, Gartner-Dataquest estimates that 50% of all chips designed today contain embedded processors, and that this number will rise to 75% by 2004. Clearly, over time, the bulk of chips are going to be designed using processors and embedded software as the most cost-effective way to create any particular functionality. Only chips for systems that differentiate themselves through having the

highest performance possible, and which can command commensurately high prices, will undertake the expensive and risky task of designing a chip where the differentiation is in hardware. The same applies at the block level too. Any part of a chip that does not get meaningful system differentiation from hardware design will be implemented in software on one of the processors on the chip.

## The Supply Chain for Software-rich Chips is Already in Place

The drive towards software-rich chips has resulted in a three-tier supply chain. At the top are the system companies such as Nokia or Cisco who ship complete systems to the people who use them. In the middle are the semiconductor companies such as Intel and Texas Instruments who actually build the chips that form the heart of any electronic system today. And then there are the companies that make the processor and peripheral blocks, often called IP (for Intellectual Property) such as ARM and MIPS. The supply chain is not completely disaggregated and some companies appear in more than one box. Intel, for example, builds processors, silicon and systems.

The word “platform” has started to come into use to describe a chip in which software, perhaps along with limited hardware changes, provides all or most of the differentiation, especially when the chip will be used in a number of different systems.



There are no hard-and-fast rules about who does what in the supply chain, but primarily the IP suppliers create processor and peripheral blocks, the semiconductor companies provide the silicon platform, probably with some basic software, and the system companies provide the bulk of the software.

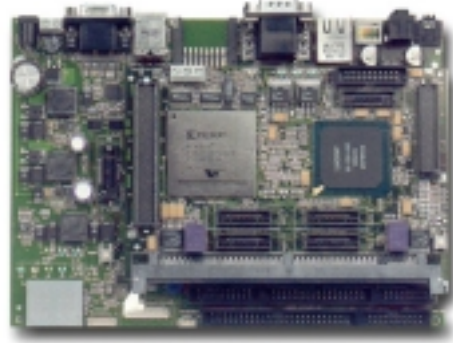
For example, a typical GSM cell-phone chip contains an ARM7TDMI processor from ARM along with an Oak DSP from DSP group. The semiconductor company will put this together into a cell-phone platform along with some basic software, perhaps including some basic functionality such as the vocoder or radio-equalization. The cell-phone manufacturer will provide the call-processing and user-interface software.

## The Embedded Software Challenge

The biggest problem with the scenario just described is that it is not usually feasible to use the actual chip to develop the embedded software that must run on it. In many cases the chip does not yet exist since the overall schedule would be unrealistically long if software development did not commence until it did. Even when the chip is available, the internals of the chip are largely invisible, making it an unattractive basis for debugging.

Typically, embedded software development is done using a “write-once, port-twice” methodology. The software is first developed on a PC, ignoring any hardware details. It is then ported to a hardware or software mockup of the chip, and finally to the chip itself.

The most common approach is to use a hardware mockup of the chip. A board is made with a processor, memory and an FPGA to hold gates that implement some of the peripherals. For a chip containing more than one processor this can be a major undertaking. The hardware board approach suffers from a number of problems. Firstly, the board forms only an approximately accurate model of the chip. For example, the cache sizes might be different from the real chip, or the bus bandwidth might be different. Problems in these areas will not show up until the real chip is available when it is too late to fix them. The second problem with hardware solutions is that it is very difficult to observe what is going on, especially in multi-processor systems where there is complex real-time interaction. And finally, hardware solutions do not scale to large software development groups who would require a prohibitively large number of boards to be manufactured and supported.



The alternative approach is to use a software mockup of the chip. An instruction-set simulator (ISS) is linked into models for the other blocks on the chip and target code is run on the ISS. Until now, ISSs have been either too slow or too inaccurate for the development to be done directly on the software mockup. If they are not fast enough, then they simply cannot be used during the edit-compile-debug phase of software development. If they are not accurate enough then they can be used to develop the software but any measurement made on the performance of the software is suspect.

## Speed and Accuracy are Both Essential

To use a mockup for developing the software directly requires both speed and accuracy. Speed is needed to make running extensive tests feasible; accuracy is needed so that the results of those tests will represent the actual behavior of the final chip.

If the speed is not there, then the edit-compile-debug cycle that is the daily grind of a software developer is just too slow. Conventional wisdom is that software developers will not tolerate their code being run at less than 10 million instructions per second (MIPS).

All embedded software has real-time constraints: the system has to guarantee to respond to certain events within a certain time. For example, the air-bag control system in a car must guarantee to deploy the air-bags within a few milliseconds of the crash sensors detecting a collision. Performance guarantees like this depend on analyzing what else may be happening on the chip at the same time, both at the hardware and software level. For modeling processors, the model used must be cycle-accurate. This means that it accurately represents the amount of time, measured in clock-cycles, that the real chip will take to execute a given code sequence. This can depend on interaction with cache memories, how well some of the performance tricks in the

processor architecture work on this particular code sequence, and what else is happening on buses on the chip that might stall the processor.

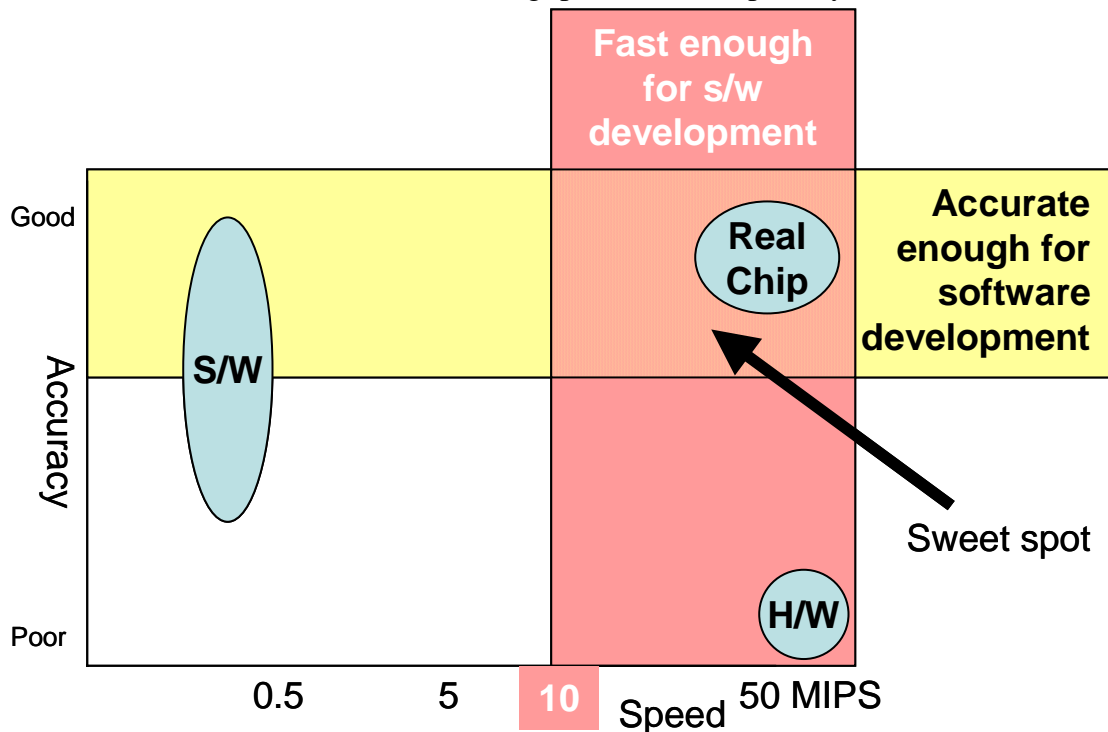
As always, the alternative to accuracy is pessimism. If it is not possible to predict accurately the response of a given microprocessor, or the time that it will take to run, then it must be over-engineered so that a faster microprocessor than necessary is used. This leaves the design open to competition from a lower-cost product that does a more accurate architectural assessment.

## VaST's Technology Delivers Speed and Accuracy

VaST provides the next generation in platform modeling, being both very fast and cycle-accurate. This allows a platform to modeled at almost real-time speeds. VaST's processor and bus models run at 30-120MIPS on an off-the-shelf PC. This is fast enough to run real applications on real operating systems, without sacrificing accuracy. This performance is over 100 times faster than any other accurate software modeling technology available. It allows full operating systems and communication protocol stacks to be exercised, and means that the software developers do not have to make compromises in how they develop software to get the performance and accuracy they need. They simply develop on the virtual platform as if it were the real chip. However, since the internals can be completely laid bare the debugging environment is immeasurably more effective.

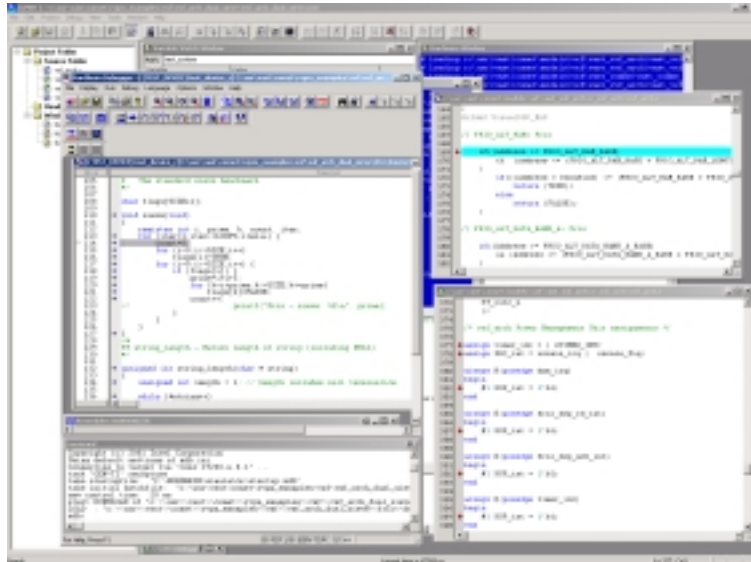
VaST's technology is solidly in the sweet spot of speed and accuracy, accurate enough to be a surrogate for the hardware from an analysis point of view, and fast enough to be a surrogate for the hardware for the software development team.

The other area where VaST's modeling provides a capability not available through either



hardware or instruction-set simulators is power modeling. VaST models processors, buses and peripherals sufficiently accurately that the power dissipated by the system running various algorithms can be calculated. This allows competing architectures and even competing software algorithms to be chosen on the basis of power dissipation.

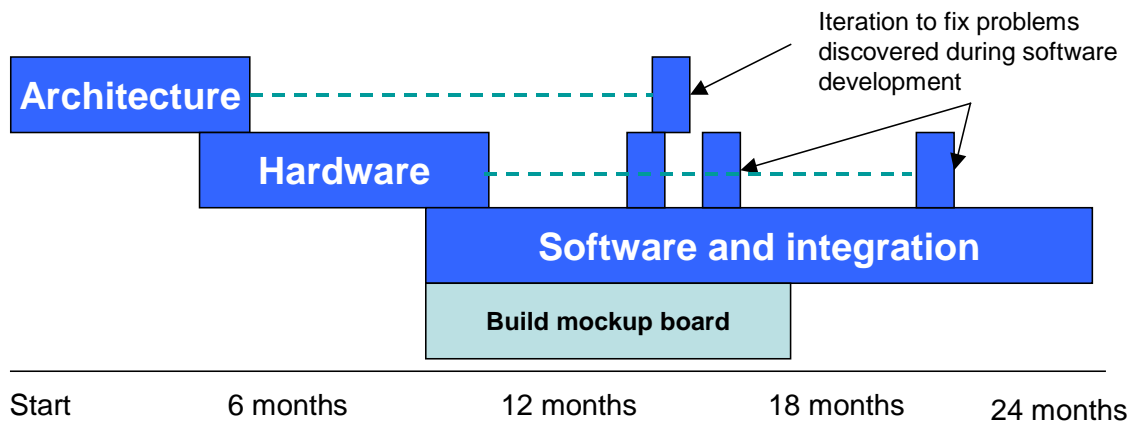
VaST's flagship product is CoMET. This is an environment for constructing and modifying models of chips. The underlying components are VaST's virtual processor models (VPMs) and VaST's unique high-performance bus modeling, which provides a transaction level interface while still preserving cycle-accuracy. Peripheral blocks are modeled at the behavioral level so that they do not become the bottleneck, although for hardware developers an RTL model of a block under development can be substituted and then exercised with the real software.



VaST has a second product called METeor for software developers. Whereas CoMET allows for creation and exploration of a platform architecture, METeor operates on a fixed platform architecture where only the software to be run on the processors can be varied. The platform to be used in METeor is, of course, created using CoMET and passed to METeor in an encapsulated form.

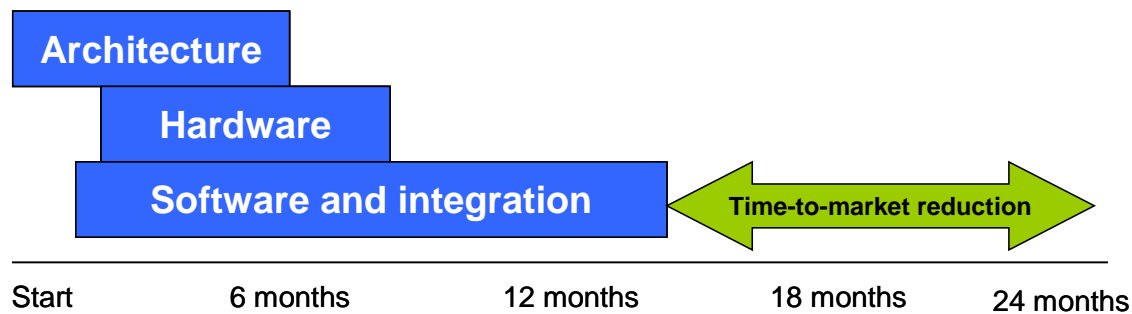
### Concurrent Development Speeds Time-to-money

The standard methodology of write-once port-twice leads to a sequential development where the architecture is finalized before either hardware or software development can start. When the hardware design is complete enough, then it is possible to build a hardware mockup and software



development can begin in earnest. This leads to the familiar pattern that the software is the critical path to delivery of the overall system. The problems are aggravated by the decisions about design tools being driven almost entirely by the needs of the hardware designers without really taking the software designers' requirements into account. This stems from a chip-centric view of system design. The architecture and hardware development are done reasonably efficiently, any mistakes showing up as a software schedule slip as the software engineers attempt to shoe-horn the software into a deficient chip implementation. Only as a last resort will the architects revisit the decisions that they consider frozen.

VaST's approach to modeling the architecture of platforms allows software development to get started once the basis architectural decisions, such as processor choices, have been made. This allows architecture, hardware development and software development to be overlapped. While it is debatable to what extent concurrent development like this reduces the overall effort, it is clear that it significantly reduces the overall development schedule. Further, the architecture is not considered complete until the software is done. In this way, by lowering the threshold to changes in the architecture, it enables convergence of the architecture on an optimal price/power/speed point.



It is worth emphasizing that nobody makes any money until a chip ships in volume, which will not happen until software development is complete. Taking the time to build a model of the architecture will do more than anything else to bring in the embedded software schedule, even if it takes manpower away from architectural exploration and hardware development. This is especially counter-intuitive in a chip company where the customer is doing software development: slipping the tapeout date to deliver a model to the customer will accelerate the time to money.

## Software Development Can Start Immediately

Software development is done using METeor together with a platform created and encapsulated by the system architects using CoMET. The platform model is accurate enough that software development that requires intimate involvement with the hardware can be undertaken. This does not just include such tasks as device-driver development and operating system porting. It also includes complex interaction between multiple processors, communication stacks and other system components that are hard to develop in isolation from the underlying hardware.

The platform model of the chip is timing-accurate, so even complex interactions such as those involving bus contention or multiple processors are handled accurately.

In addition, using the power modeling capabilities in METeor, it is possible to measure the implications of software decisions on peak power, battery life and other important power attributes.

Despite all the best efforts of the architects, the reality of software-rich chip development is that many architectural problems will be found by the software engineers. It is never possible to think of everything in advance. Only when the real software is finally written is it possible to assess the system performance completely. The type of problems that may come to light at this stage are: bus-contention issues where a bus-master is blocked unacceptably long from communicating; cache-sizing issues where unacceptable cache-miss percentages degrade the overall performance of the system to an unacceptable level; excessive power consumption, possibly as a result of excessive cache line-refills; capacity issues such as a bus with inadequate bandwidth to transfer the data load, or a processor that is unable to complete its assigned workload to schedule.

In a traditional serial development where architecture precedes hardware design, which in turn precedes software design, this type of problem is a disaster. It necessitates re-work of parts of the design that were considered completed. To make things worse, the team that needs to do the re-work may well be off on its next project and getting their attention can be difficult.

## Summary

Moore's law and the compelling semiconductor economics that it drives are driving the growth of software-rich chips, systems implemented using increasingly large quantities of embedded software for their differentiation.

VaST's unique simulation technologies allow architectural exploration and software development to take place using a software model of the chip. This is the only technology that is both fast enough to allow software development to be done on such a model, and accurate enough that measurements of performance and power can be made from the model that reflect how the eventual system will behave.

This technology enables the transition from a chip-centric view of system design, where the software is little more than an afterthought, to a software-centric view of system design, where chip design is regarded as the creation of an efficient platform for running the software.